Hw4 report

姓名：傅敬倫　　系級：電機四　　學號：b06505011

Problem 1　　no collaboration

(1)

```
Convnet(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
)
```

上圖是我的 model 架構，跟助教 ppt（p.26）上是一樣的。其中我是沒有用到 MLP 的，在訓練中我是用 Sampler 去取樣，以下為我的 Sampler:

```python
class CategoriesSampler():

    def __init__(self, label, n_batch, n_cls, n_per):
        self.n_batch = n_batch
        self.n_cls = n_cls
        self.n_per = n_per

        label = np.array(label)
        self.m_ind = []
        for i in range(max(label) + 1):
            ind = np.argwhere(label == i).reshape(-1)
            ind = torch.from_numpy(ind)
            self.m_ind.append(ind)

    def __len__(self):
        return self.n_batch

    def __iter__(self):
        for i_batch in range(self.n_batch):
            batch = []
            classes = torch.randperm(len(self.m_ind))[:self.n_cls]
            for c in classes:
                l = self.m_ind[c]
                pos = torch.randperm(len(l))[:self.n_per]
                batch.append(l[pos])
            batch = torch.stack(batch).t().reshape(-1)
            yield batch
```

Detail of training:

epoch	Distance function	way
15	Euclidean	15

shot	query	Episode(per epoch)
1	15	1000

Model learning rate	optimizer
0.001	Adam

The accuracy on validation set under 5-way 1-shot setting : $46.93 \pm 0.85\%$

(2)

方法	Validation accuracy
Euclidean	$43.78 \pm 0.80\%$
Cosine	$30.75 \pm 0.58\%$
Parametric_version1	$45.74 \pm 0.84\%$
Parametric_version2	$44.71 \pm 0.78\%$

Common details of training:

epoch	way	shot	query	optimizer
5	5	1	15	Adam

Special details of Euclidean:
Model 使用助教給的 Convnet（第一小題），沒有使用 MLP
Model learning rate = 0.001

Special details of Cosine Similarity:
Model 使用助教給的 Convnet（第一小題），沒有使用 MLP
Model learning rate = 0.0001

(2)

Special details of Parametric function:
Model 使用助教給的 Convnet（第一小題），沒有使用 MLP
Model learning rate = 0.001
在 parametric function 我有使用了兩種方法：
（1） 將 latent space(1600)和 proto(1600)直接 concatenate(3200)作為 input
（2） 將 latent space(1600)減 proto(1600)作為 input

定義自己的 Parametric model:
（1） 輸入大小： N * 3200 , 輸出大小：N * 1 （預測距離）

```
distance(
  (main): Sequential(
    (0): Linear(in_features=3200, out_features=200, bias=True)
    (1): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=200, out_features=1, bias=True)
    (4): Tanh()
  )
)
```

（2）輸入大小： N * 1600 , 輸出大小：N * 1 （預測距離）

```
distance(
  (main): Sequential(
    (0): Linear(in_features=1600, out_features=200, bias=True)
    (1): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=200, out_features=1, bias=True)
    (4): Tanh()
  )
)
```

在這兩種方法中的 D（parametric model）的 optimizer 為 Adam，learning rate 皆為 0.003

(3)

名稱	Validation accuracy （under 5-wat K-shot）
K=1	35.29%
K=5	53.32%
K=10	63.32%

Details of training:

名稱	epoch	Distance function	way
K=1	5	Euclidean	5
K=5	5	Euclidean	5
K=10	5	Euclidean	5

名稱	shot	query	Episode (per epoch)
K=1	1	15	1000
K=5	5	15	1000
K=10	10	15	1000

名稱	Model learning rate	optimizer
K=1	0.001	Adam
K=5	0.001	Adam
K=10	0.001	Adam

Problem 2　　no collaboration

(1)

```
Convnet(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
)

Generator(
  (main): Sequential(
    (0): Linear(in_features=1600, out_features=800, bias=True)
    (1): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=800, out_features=800, bias=True)
    (4): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=800, out_features=400, bias=True)
    (7): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): Linear(in_features=400, out_features=400, bias=True)
    (10): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): Linear(in_features=400, out_features=1600, bias=True)
    (13): Tanh()
  )
)
```

上圖為我的 model 和 Hallucinator 架構，一樣沒有使用 MLP。Sampler 和 Problem1 是用一樣的。Hallucinator 的 input 則是把 model 出來的 latent space 每個 class 隨機抽取出一個並且複製 M 次，加上 M 個 noise。因此最後會有 way * shot 個 real data 和 way * M 個 hallucination data，在每個 class 中去做平均當作 proto。在此題中的 distance function 都使用 Euclidean。

Detail of training:

epoch	Distance function	way
15	Euclidean	15

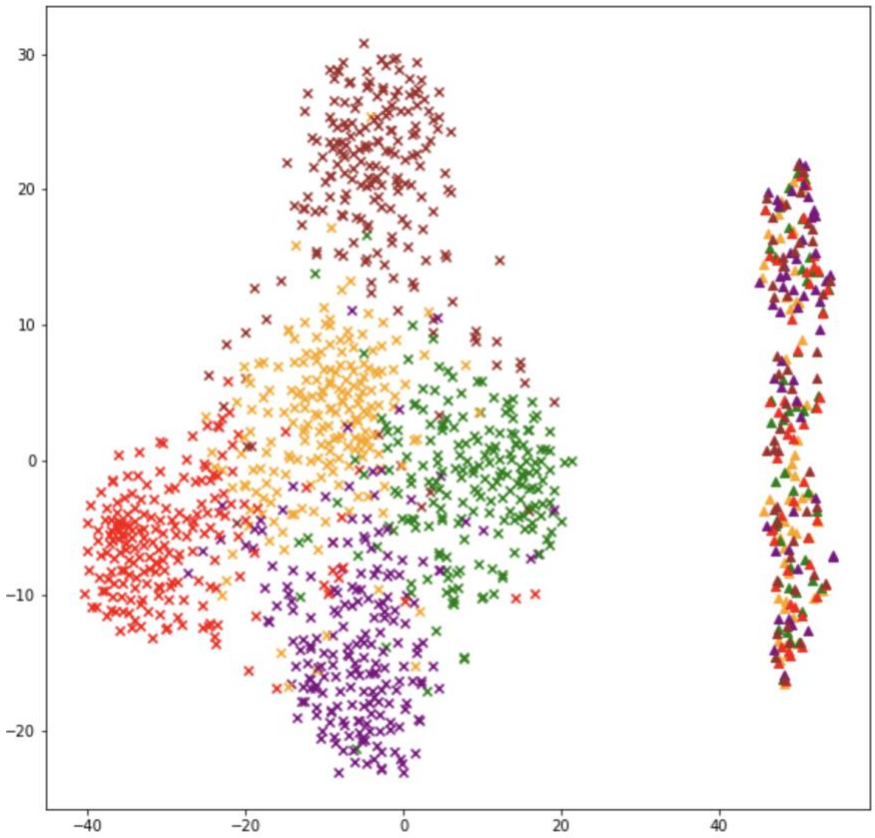
shot	query	Episode(per epoch)
1	15	1000

Model learning rate	G learning rate	Optimizer(model,G)
0.001	0.0005	Adam

我使用 M=100

The accuracy on validation set under 5-way 1-shot setting : $47.34 \pm 0.83\%$

(2)



可以發現叉叉（real data）已經被分類的蠻好的，可以看出有五大地區。但在三角形（Hallucination data）是沒有被分類好的，而且散步的地方離 real data 有一段距離。因此能改進的地方是能讓 real 和 hallucination 能靠近一點，並且不同類別也可以被分開。

(3)

名稱	epoch	M	way
M=10	15	10	15
M=50	15	50	15
M=100	15	100	15

名稱	shot	query	Optimizer (under model, Hallucinator)
M=10	1	15	Adam
M=50	1	15	Adam
M=100	1	15	Adam

名稱	Model learning	Hallucinator learning rate	Episode(per epoch)
M=10	0.001	0.0005	1000
M=50	0.001	0.0005	1000
M=100	0.001	0.0005	1000

名稱	Validation accuracy （under 5-wat 1-shot M-augmentation）
M=10	46.61%
M=50	47.04%
M=100	47.34%

以實驗結果來說，當 M 越大也就是 Hallucination data 越多，在 validation set 的 accuracy 越高，也就是算出平均的 proto 越好。以直觀來講這樣的結果並不意外，因為原本只有 one shot，經過 model 的 proto 可能不 general，但是如果加上越多自己製造的 data（也不能離 real data 太遠），算出的 proto 也會越接近真實。但以時間來說，M 越大花的時間也越多。

(4)

在實作 hallucination model 時，我一開始在 hallucinator 的部分是直接拿 Gan 的 generator，但 train 的效果超級爛。應該是因為用 convolution 去 generate 可能太複雜了，而且 Hallucinator 的 input 已經是經過 cnn 的 feature，因此直接用 linear 去堆疊就好了。最後，我的 Hallucinator 就是用 linear layers。在實驗時，model 和 Hallucinator 的 learning rate 是要花時間去試，太大的話 loss 基本上都沒什麼變化。

Problem3　　no collaboration

1.

```
Convnet(
  (encoder): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
  )
)

Generator(
  (main): Sequential(
    (0): Linear(in_features=1600, out_features=800, bias=True)
    (1): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Linear(in_features=800, out_features=800, bias=True)
    (4): BatchNorm1d(800, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=800, out_features=400, bias=True)
    (7): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): Linear(in_features=400, out_features=400, bias=True)
    (10): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): Linear(in_features=400, out_features=1600, bias=True)
    (13): Tanh()
  )
)
```

在這題中我使用的 model 和 hallucinator 和第二小題是一模一樣的，差別在我使用 loss 的時候我多加了一個 MSELoss，因為我想讓 real data 和 hallucinator data 兩個距離更近一點，我這題是使用 M=100 來 train

Details of training:

epoch	Distance function	way
15	Euclidean	15

shot	query	Episode(per epoch)
1	15	1000

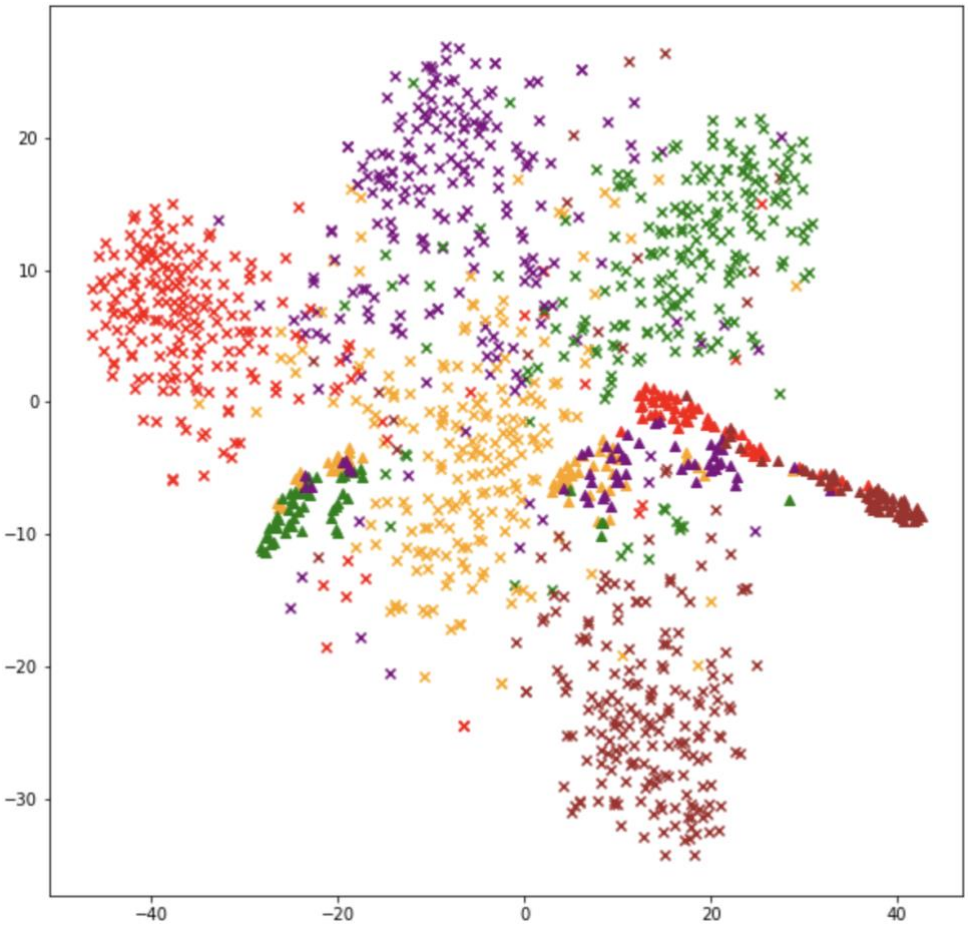
Model learning rate	Hallucinator learning rate	optimizer
0.001	0.0005	Adam

```
domainclass(
  (shared_encoder_pred_domain): Sequential(
    (0): Linear(in_features=1600, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

另外，我也有常嘗試利用 Discriminator 來讓 real data 和 hallucination data 更加接近，train 的方法為第一次先讓 discriminator 可以正確分出 real 和 hallucination，在第二次更新時利用減法來讓 discriminator 搞混，使用想法和 DSN 很接近，但最後我 train 出來的結果並沒有比第二小題好，因此可能是 learning rate 沒調整好，或是還需要更久的 epoch

最後我是使用 M=100，且 train 的時候多加 MSELoss 的 model
The accuracy on validation set under 5-way 1-shot setting : $48.89 \pm 0.84\%$

(2)



可以發現在 real data（叉叉）的部分也都有明顯的分類成五大部分。在 hallucination data（三角形）的部分只能比較看出分成兩坨，與 real data 的距離也蠻靠近的。

(3)

在第三題 improve 中，會比第二題正確率來得高是因為我有讓 hallucinator 產生出的 data 與 real data 更為相似。可以從兩小題的 tsne 發現第三題的 hallucination data 與 real data 分佈更加集中，雖然單看 hallucination data 可能沒有到分類的非常好，但確實比第二題好很多，因此算出的 proto 也會更準確。